LAPORAN PRAKTIKUM
PEMROGRAMAN BERORIENTASI OBJEK

MODUL VII

RELASI

Oleh:

Ramdhan Wijaya

2211102208

IF-10-M

S1 TEKNIK INFORMATIKA

INSTITUT TEKNOLOGI TELKOM PURWOKERTO

2024

I. DASAR TEORI

Agregasi

Agregasi adalah relasi dua/banyak objek yang salah satu objek tersebut sebagai wadah objek objek dari class lain. Agregasi juga merupakan bentuk hubungan yang lebih khusus dari Assosiasi dimana sebuah object memiliki lifecycle nya sendiri tapi dengan kepemilikan dan class child tidak dapat memiliki class parentnya. Relasinya biasa di sebut relasi “has-a”.Dalam diagram UML, relasi agregat ini digambarkan dengan simbol diamond. Simbol ini menunjukkan adanya inklusi struktural sebuah objek terhadap objek yang lain yang diwujudkan dengan cara membuat kelas agregat yang memiliki atribut yang bertipe kelas penyusun.

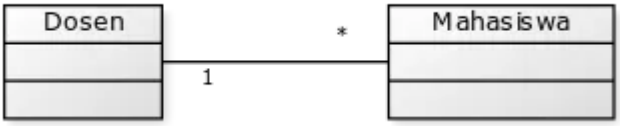
Jika hubungan asosiasi adalah saling menggunakan, di sini hubungan yang terjadi adalah memiliki. Dilihat pada gambar kelas diagram di atas, bahwa kelas Jurusan memiliki kelas Mahasiswa sebagai variable nya. Meskipun kelas Jurusan memiliki kelas Mahasiswa, namun kedua kelas tersebut dapat dibuat secara independen.

1. gambar relasi agregasi

Asosiasi

Assosiasi adalah hubungan antara object. Relasi assosiasi biasa disebut “is a” relationship. Assosiasi berarti bahwa sebuah object “menggunakan” object yang lain. Assosiasi adalah sebuah relasi dimana semua object memiliki lifecycle nya sendiri dan tidak ada yang bertindak sebagai owner.

Contoh Teacher dan Student. Banyak Student dapat berasosiasi dengan satu Teacher dan satu Student dapat berasosiasi dengan banyak Teacher. Object keduanya bisa dibuat dan dihancurkan secara mandiri tanpa berpengaruh pada eksistensi object lain.



2.gambar relasi asosiasi

Dependency

dependency injection adalah ketergantungan antara satu instansi class dengan instansi class yang lain. Dependency sejujurnya memiliki arti yang luas dalam pemograman itu sendiri. Dependency dapat juga berarti suatu library yang digunakan dalam suatu aplikasi, dapat juga berarti ketergantungan antara satu aplikasi dengan aplikasi yang lain. Jadi jangan terpaku terhadap istiliah dependency dalam dependency injection saja. Sebuah aplikasi tentunya terdiri dari berbagai macam instansi dari class.

Mengenal Dependency Inversion

Kondisi saat ini, masing-masing class membuat instansinya dependency-nya masing-masing. Dalam artian masing-masing class memiliki ketergantungan yang kuat dengan class yang lain. Terus kenapa kalau begitu? Apakah itu merupakan suatu hal yang buruk?

Dalam kasus-kasus pada umumnya, ketergantungan yang kuat (tighly-coupled) antara satu class dengan class yang lain itu merupakan hal yang buruk dan dihindari. Ketergantungan tersebut membuat class yang bergantung pada class lain tersebut, ringkih terhadap perubahan yang terjadi pada class lain tersebut. Belum lagi jika terdapat banyak class yang bergantung pada sebuah class, maka kamu harus menyesuaikan semua class yang bergantung pada satu class tersebut.

3.gambaran relasi dependency

II. GUIDED

IbuAnak.cpp

```
#include <string.h>
#include <iostream>
#include <conio.h>
using namespace std;

class Manusia
{
private:
    char nama[20];
    int umur;
    Manusia *anak;
    // Letak asosiasi, dimana kelas anak menjadi variabel
    Manusia *ibu;

public:
    Manusia(char *nama, int umur);
    Manusia(char *nama, int umur, Manusia &ibu);
    void cetak();
    void adopsi(Manusia &anak);
    // Relasi yang menunjukkan asosiasi
};

Manusia::Manusia(char *nm, int umr)
{
    strcpy(nama, nm);
    umur = umr;
    ibu = NULL;
    anak = NULL;
}

Manusia::Manusia(char *nm, int umr, Manusia &ibu_angkat)
{
    strcpy(nama, nm);
    umur = umr;
    anak = NULL;
    ibu = &ibu_angkat;
    ibu_angkat.anak = this;
}

void Manusia::cetak()
{
    cout << endl << "- Data Pribadi -" << endl;
```

```
    cout << "Nama : " << nama << endl;
    cout << "Umur : " << umur << endl;
    if (ibu)
        cout << "Nama ibu : " << ibu->nama << endl;
    if (anak)
        cout << "Nama anak : " << anak->nama << endl;
}

void Manusia::adopsi(Manusia &anak_angkat)
{
    anak = &anak_angkat;
    anak_angkat.ibu = this;
}

int main()
{
    Manusia ibu1("Budi", 30);
    Manusia anak1("Ani", 4);
    Manusia ibu2("Diana", 40);
    Manusia anak2("Andi", 5, ibu2);
    // Relasi antara Manusia dengan Manusia
    cout << endl << "=========================" << endl;
    ibu1.cetak();
    anak1.cetak();
    cout << endl << "=========================" << endl;
    ibu1.adopsi(anak1);
    ibu1.cetak();
    anak1.cetak();
    cout << endl << "=========================" << endl;
    ibu2.cetak();
    anak2.cetak();
    cout << endl;
    getch();
    return 0;
}
```

Output :

```
========================

- Data Pribadi -
Nama : Budi
Umur : 30

- Data Pribadi -
Nama : Ani
Umur : 4

========================

- Data Pribadi -
Nama : Budi
Umur : 30
Nama anak : Ani

- Data Pribadi -
Nama : Ani
Umur : 4
Nama ibu : Budi

========================

- Data Pribadi -
Nama : Diana
Umur : 40
Nama anak : Andi

- Data Pribadi -
Nama : Andi
Umur : 5
Nama ibu : Diana
```

Keterangan :
pada program IbuAnak.cpp diatas adalah membuat program relasi asosiasi karena objek ibu dan anak Ketika dihilangkan tidak akan berpengaruh karena tidak ada ketergantungan, dan itulah kenapa disebut hubungan “is a” yang maksudnya class ibu adalah class anak juga jika dihubungkan dengan program diatas.

Relasi.cpp

```
#include <iostream>
#include <conio.h>
#include <string.h>
using namespace std;

class Pegawai // Class Pegawai
{
private:
    char NIP[5];
    char nama[20];

public:
    Pegawai();
    Pegawai(char *nip, char *nm);
    void tampilPeg();
};
Pegawai::Pegawai()
{
    cout << "Konstruktor pegawai dijalankan..." << endl;
}
Pegawai::Pegawai(char *nip, char *nm)
{
    strcpy(NIP, nip);
    strcpy(nama, nm);
}
void Pegawai::tampilPeg()
{
    cout << "NIP : " << NIP << " ,Nama : " << nama << endl;
}
class Perusahaan // Class Perusahaan
{
private:
    int counter;
    char namaPer[20];
    Pegawai peg[3]; // agregasi antara pegawai dan
perusahaan
public:
    Perusahaan(char *nm);
    void insertPegawai(Pegawai p);
    // Relasi yang menunjukkan agregasi
    void tampilPer();
};
Perusahaan::Perusahaan(char *nm)
```

```
{
    strcpy(namaPer, nm);
    counter = 0;
    cout << "Konstruktor perusahaan dijalankan..." << endl;
}
void Perusahaan::insertPegawai(Pegawai p)
{
    peg[counter] = p;
    counter++;
}
void Perusahaan::tampilPer()
{
    cout << "Perusahaan " << namaPer << " memiliki pegawai
:" << endl;
    for (int i = 0; i < counter; i++)
    {
        peg[i].tampilPeg();
    }
}

int main()
{
    Perusahaan Per("Nusantara Jaya");
    Pegawai Peg1("P001", "Rudi");
    Pegawai Peg2("P002", "Toni");
    Pegawai Peg3("P003", "Ani");
    // Relasi antara Class Pegawai dengan Class Perusahaan
    Per.insertPegawai(Peg1);
    Per.insertPegawai(Peg2);
    Per.insertPegawai(Peg3);
    cout << endl;
    Per.tampilPer();
    getch();
    return 0;
}
```

Output :

```
Perusahaan Nusantara Jaya memilik
i pegawai :
NIP : P001 ,Nama : Rudi
NIP : P002 ,Nama : Toni
NIP : P003 ,Nama : Ani
```

Keterangan :
Program diatas adalah program relasi agregasi karena terdapat dua kelas dan salah satu kelasnya adalah kelas penyusun dan satunya adalah kelas yang memiliki. Hubungan pada program diatas adalah memiliki atau disebut “has a” yang Dimana class Perusahaan memiliki class pegawai di deklarasinya. Jika class pegawai dihapus maka tidak akan terjadi apa apa, namun Ketika di run maka programnya hanya mengeprint apa yang ada pada class Perusahaan. dan itulah kenapa class agregasi disebut class yang memiliki tetapi hubungannya tidak wajib seperti dependency, seperti yang dijelaskan tadi jika class pegawai dihapus maka class Perusahaan tetap bisa berjalan.

Guided 1

IbuAnak.java

```
package guided1;

/**
 *
 * @author rama
 * 2211102208
 */
public class IbuAnak {

    public static void main(String[] args) {
        Manusia ibu1 = new Manusia("Budi", 30);
        Manusia anak1 = new Manusia("Ani", 4);
        Manusia ibu2 = new Manusia("Diana", 40);
        Manusia anak2 = new Manusia("Andi", 5, ibu2);
//Relasi antara Manusia dengan Manusia

System.out.println("========================\n");
        ibu1.cetak();
        anak1.cetak();

System.out.println("========================\n");
```

```
        ibu1.adopsi(anak1);
        ibu1.cetak();
        anak1.cetak();

System.out.println("========================\n");
        ibu2.cetak();
        anak2.cetak();
    }
}
```

Manusia.cpp

```
package guided1;

/**
 *
 * @author rama
 * 2211102208
 */
class Manusia {

    private String nama;
    private int umur;
    private Manusia ibu;
    private Manusia anak;
//Letak asosiasi, kelas anak jadi variabel

    public Manusia() {
    }

    public Manusia(String nm, int umr) {
        nama = nm;
        umur = umr;
        ibu = new Manusia();
        anak = new Manusia();
        ibu = null;
        anak = null;
    }

    public Manusia(String nm, int umr, Manusia ibu_angkat) {
```

```
        nama = nm;
        umur = umr;
        ibu = new Manusia();
        anak = new Manusia();
        ibu = ibu_angkat;
        ibu_angkat.anak = this;
    }
//Relasi yang menunjukkan asosiasi

    public void adopsi(Manusia anak_angkat) {
        anak = anak_angkat;
        anak_angkat.ibu = this;
    }

    public void cetak() {
        System.out.println("\n- Data Pribadi -");
        System.out.println("Nama : " + nama);
        System.out.println("Umur : " + umur);
        if (ibu != null) {
            System.out.println("Nama ibu : " + ibu.nama);
        } else if (anak != null) {
            System.out.println("Nama anak : " + anak.nama);
        }

    }
}
```

Output :

```
========================


- Data Pribadi -
Nama : Budi
Umur : 30

- Data Pribadi -
Nama : Ani
Umur : 4
========================


- Data Pribadi -
Nama : Budi
Umur : 30
Nama anak : Ani

- Data Pribadi -
Nama : Ani
Umur : 4
Nama ibu : Budi
========================


- Data Pribadi -
Nama : Diana
Umur : 40
Nama anak : Andi

- Data Pribadi -
Nama : Andi
Umur : 5
Nama ibu : Diana
```

Keterangan :
Program diatas sama seperti program cpp sebelumnya hanya saja ini menggunakan bahasa pemograman java, yang menjelaskan relasi asosiasi Dimana object anak juga adalah objek ibu hanya saja ada method yang dapat membuat sang ibu mengadopsi anak. Realsi ini disebut relasi asosiasi/"is a".

Guided 2

Pegawai.java

```
package guided2;
```

```
/**
 *
 * @author rama
 * 2211102208
 */
// Class Pegawai
class Pegawai {

    private String nama;
    private String NIP;

    public Pegawai() {
        System.out.println("Konstruktor pegawai dijalankan...");
}
public Pegawai(String NIP, String nama) {
        this.NIP = NIP;
        this.nama = nama;
        System.out.println("Konstruktor pegawai dijalankan...");
}
public void tampilPeg() {
        System.out.println("NIP : " + NIP + " ,Nama : " +
nama);
    }
}
```

Perusahaan.java

```
package guided2;

/**
 *
 * @author rama
 * 2211102208
 */
// Class Perusahaan
class Perusahaan {

    private final String namaPer;
```

```
    private final Pegawai peg[]; //Agregasi antara pegawai dan
perusahaan
    private int counter;

    public Perusahaan(String namaPer) {
        this.namaPer = namaPer;
        counter = 0;
        peg = new Pegawai[3];
        System.out.println("Konstruktor perusahaan
dijalankan...");
}
//Relasi agregasi antara pegawai dan perusahaan
public void insertPegawai(Pegawai p) {
        peg[counter] = p;
        counter++;
    }

    public void tampilPer() {
        System.out.println("Perusahaan " + namaPer + "
memiliki pegawai: ");
for (int i = 0; i < counter; i++) {
            peg[i].tampilPeg();
        }
    }
}
```

Relasi.java

```
package guided2;

/**
 *
 * @author rama
 * 2211102208
 */
public class Relasi {

    public static void main(String[] args) {
        Perusahaan Per = new Perusahaan("Nusantara Jaya");
        Pegawai Peg1, Peg2, Peg3;
```

```
        Peg1 = new Pegawai("P001", "Rudi");
        Peg2 = new Pegawai("P002", "Toni");
        Peg3 = new Pegawai("P003", "Ani");
//Relasi antara Class Pegawai dengan Class Perusahaan
        Per.insertPegawai(Peg1);
        Per.insertPegawai(Peg2);
        Per.insertPegawai(Peg3);
        System.out.println();
        Per.tampilPer();
    }
}
```

Output :

```
Konstruktor perusahaan dijalankan...
Konstruktor pegawai dijalankan...
Konstruktor pegawai dijalankan...
Konstruktor pegawai dijalankan...

Perusahaan Nusantara Jaya memiliki pegawai:
NIP : P001 ,Nama : Rudi
NIP : P002 ,Nama : Toni
NIP : P003 ,Nama : Ani
BUILD SUCCESSFUL (total time: 0 seconds)
```

Keterangan :
Program diatas sama seperti program relasi.cpp diatas namun di convert ke bahasa pemograman java, yang Dimana menjelaskan hubungan agregasi yang diantara class penyusun dan class agregasi yang memiliki class class penyusunnya di dalam.

Question: Amati Relasi.java dan Relasi.cpp ! Apakah ada perbedaan antara kedua code tersebut ? Jelaskan dalam laporan Anda !

Answer: tidak ada karena sama sama menjelaskan tentang agregasi tentang Perusahaan dan pegawai yang ditentukan

memiliki maksimal 3 pegawai. Dalam pengkodean pun seperti code yang di program c++ hanya di convert di bahasa java saja.

III. UNGUIDED

Source code

Titik.java

```
package Unguided1;

/**
 *
 * @author rama
 * 2211102208
 */

public class Titik {

  private double x;
  private double y;

  public double getX() {
    return x;
  }

  public double getY() {
    return y;
  }

  public Titik(double X, double Y) {
    x = X;
    y = Y;
  }

  public void tampil() {
    System.out.println("Point\t\t\t: [" + x + ", " + y + "]");
  }

  public double hitungJarak(Titik t2) {
    return Math.sqrt(Math.pow(t2.getX() - x, 2) +
Math.pow(t2.getY() - y, 2));
  }

}
```

Persegi.java

```
/**
 *
 * @author rama
 * 2211102208
 */
public class Persegi extends Titik {

  private double sisiPSG;

  public Persegi(double x1, double y1, double x2, double y2)
{
    super(x1, y1);
    Titik t2 = new Titik(x2, y2);
    sisiPSG = hitungJarak(t2);
  }

  public double hitungLuas() {
    return Math.pow(sisiPSG, 2);
  }

  @Override
  public void tampil() {
    System.out.println("== DATA Persegi ==");
    super.tampil();
    System.out.println("Sisi Persegi\t\t: " + sisiPSG);
    System.out.println("Luas Persegi\t\t: " + hitungLuas());
  }

}
```

SgtSamaSisi.java

```
/**
 *
 * @author rama
 * 2211102208
 */
public class SgtSamaSisi extends Titik {
```

```
  private double sisiSGT;

  public SgtSamaSisi(double x1, double y1, double x2, double
y2) {
    super(x1, y1);
    Titik t2 = new Titik(x2, y2);
    sisiSGT = hitungJarak(t2);
  }

  public double hitungLuas() {
    // tinggi segitiga dengan rumus Pytagoras
    double tinggiSegitiga = Math.sqrt(Math.pow(sisiSGT, 2) -
Math.pow((sisiSGT / 2), 2));

    return sisiSGT * tinggiSegitiga / 2;
  }

  @Override
  public void tampil() {
    System.out.println("== DATA Segitiga ==");
    super.tampil();
    System.out.println("Sisi Segitiga\t\t: " + sisiSGT);
    System.out.println("Luas Segitiga\t\t: " + hitungLuas());
  }

}
```

Limas.java

```
/**
 *
 * @author rama
 * 2211102208
 */
public class Limas {

  private SgtSamaSisi sgtSamaSisi;
  private Persegi persegi;
```

```
  public Limas(double Sx1, double Sy1, double Sx2, double
Sy2, double Px1, double Py1, double Px2, double Py2) {

    sgtSamaSisi = new SgtSamaSisi(Sx1, Sy1, Sx2, Sy2);
    persegi = new Persegi(Px1, Py1, Px2, Py2);

  }

  public double hitungLuas() {
    return (4 * sgtSamaSisi.hitungLuas()) +
persegi.hitungLuas();
  }

  public void tampil() {
    sgtSamaSisi.tampil();
    System.out.println();

    persegi.tampil();
    System.out.println();

    System.out.println("Luas Limas\t\t: " + hitungLuas());
  }



}
```

Relasi.java

```
/**
 *
 * @author rama
 * 2211102208
 */
public class Relasi{
        public static void main(String args[]) {
          Limas P = new Limas(1.0,2.0,5.0,2.0,2.0,3.0,2.0,7.0);
          P.rumus();
        }
    }
```

```
   class Limas{
      double Sx1; double Sy1; double Sx2; double Sy2; double
Px1; double Py1; double Px2; double Py2;
      Limas(double Sx1,double Sy1, double Sx2, double Sy2,
double Px1, double Py1, double Px2, double Py2){
         this.Sx1 = Sx1;
         this.Sy1 = Sy1;
         this.Sx2 = Sx2;
         this.Sy2 = Sy2;
         this.Px1 = Px1;
         this.Px2 = Px2;
         this.Py1 = Py1;
         this.Py2 = Py2;
      }
      void rumus(){
         double luas_Segitiga, luas_Persegi, luas_Limas;
         System.out.println("=== Data Segitiga ===");
         System.out.println("point\t: " + this.Sx1 + ", " +
this.Sy1);
         System.out.println("Sisi Segitiga\t: " + (this.Sy1 +
this.Sy2));
         luas_Segitiga = 0.5 * (this.Sy1 + this.Sy2) *
Math.sqrt(3);
         System.out.println("Luas Segitiga\t: " +
luas_Segitiga);


         System.out.println("\n=== Data Persegi ===");
         System.out.println("point\t: " + this.Px1 + ", " +
this.Py1);
         System.out.println("Sisi Persegi\t: " + (this.Px1 +
this.Px2));
         luas_Persegi = (this.Px1 + this.Px2) * (this.Px1 +
this.Px2);
         System.out.println("Luas Persegi\t: " + luas_Persegi);

         luas_Limas = (4 * luas_Segitiga) + luas_Persegi;
         System.out.println("\n\nLuas Limas : " + luas_Limas);
```

```
    }
}
```

Output :

```
run:
=== Data Segitiga ===
point   : 1.0, 2.0
Sisi Segitiga   : 4.0
Luas Segitiga   : 3.4641016151377544

=== Data Persegi ===
point   : 2.0, 3.0
Sisi Persegi    : 4.0
Luas Persegi    : 16.0


Luas Limas : 29.856406460551018
```

Keterangan :
Program diatas adalah sebuah program dengan hubungan/relasi di setiap kelasnya, class titik memiliki relasi asosiasi dengan persegi dan SgtSamasisi. Class limas memiliki relasi dependency dengan class persegi dan SgtSamasisi Dimana jika salah satu class yang dapat berdiri sendiri dihapus maka class yang memiliki ketergantungan akan bermasalah, dalam kasus ini class yang mempunyai ketergantungan adalah class limas.